WEST Search History

DATE: Wednesday, July 02, 2003

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L17	5974251.pn.	1	L17
DB=EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L16	L14	23	L16
DB=JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L15	L14 and instrument$	0	L15
L14	(generat$ stub)	38	L14
DB=USPT,PGPB; PLUR=YES; OP=ADJ			
L13	L12 and instrument$	1	L13
L12	5313616.pn. and (generat$ stub)	1	L12
L11	L10 and (class or function)	22	L11
L10	L2 and (generat$ stub)	22	L10
L9	L1 and (generat$ stub)	25	L9
L8	L1 and (generat$ stub) near2 (source or program)	1	L8
L7	L2 and (generat$ stub) near2 (source or program)	1	L7
L6	L2 and (generat$ stub) near (source or program)	0	L6
DB=USPT; PLUR=YES; OP=ADJ			
L5	L4	11	L5
DB=USPT,PGPB; PLUR=YES; OP=ADJ			
L4	L3 and generat$ near2 stub	42	L4
L3	L2 and identi$ near2 (function or method or subroutine)	2523	L3
L2	L1 and compil$	9509	L2
L1	instrument$	331276	L1

END OF SEARCH HISTORY

Search Results - Record(s) 1 through 20 of 23 returned.

☐ 1. Document ID: EP 643349 A1

L16: Entry 1 of 23 File: EPAB Mar 15, 1995

PUB-NO: EP000643349A1
DOCUMENT-IDENTIFIER: EP 643349 A1
TITLE: Client-side stub interpreter.

PUBN-DATE: March 15, 1995

INVENTOR-INFORMATION:

NAME	COUNTRY
KESSLER, PETER B	US
HAMILTON, GRAHAM	US
GIBBONS, JONATHAN J	US

INT-CL (IPC): G06 F 9/46
EUR-CL (EPC): G06F009/44; G06F009/46, G06F009/46

ABSTRACT:

CHG DATE=19990617 STATUS=O> The present invention provides an elegant and compact way to provide mechanisms for invocation of objects by client applications and for argument passing between client applications and object implementations, which reduce the memory space required for the client-side stubs, without the client application or the operating system knowing the details of how these mechanisms work. Moreover, these mechanisms functions in a distributed computer environment with similar ease and efficiency, where client applications may be on one computer node and object implementations on another. Additionally the invention is independent of the particular C++ compiler used for generation of the stub code. The mechanism used to reduce this memory space comprises a stub generator (called ``CONTOCC''), a data base of client-side stub description files and a stub-interpreter which knows how to read these client-side stub description files. CONTOCC reads interface definition language (``IDL'') files and generates corresponding C++ files. CONTOCC has the ability to read the IDL data and generate either normal C++ stub files or the special client-side stub interpreter files described in more detail below.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Clip Img | Image

☐ 2. Document ID: EP 501610 A2

L16: Entry 2 of 23 File: EPAB Sep 2, 1992

PUB-NO: EP000501610A2
DOCUMENT-IDENTIFIER: EP 501610 A2
TITLE: Object oriented distributed computing system.

PUBN-DATE: September 2, 1992

INVENTOR-INFORMATION:

NAME	COUNTRY
WALDO, JAMES A	US
ARNOLD, KENNETH C	US
ERDOS, MARLENA	US
ROBINSON, DOUGLAS B	US
HOFFMAN, D JEFFREY	US
SMITH, LAMAR D	US
SHOWMAN, PETER S	US
CANNON, MICHAEL J	US
SEABORNE, ANDREW F	GB
MCBRIDE, BRIAN W	US
HARRISON, BRIAN D	US

INT-CL (IPC): C08G 59/62; G06F 9/44; G06F 15/16
EUR-CL (EPC): G06F009/46

ABSTRACT:

CHG DATE=19990617 STATUS=O> An object oriented distributed computing system (11) is provided. Processing means (15) call a location service (21) within automatically generated stubs in response to a request for a service provided by a particular object (13). The location service (21) is automatically called on behalf of the requester to locate the target object (13) when the request is issued. Multiple Object Managers (23) reflecting multiple Object Models are permitted in the system (11). Programmers and users do not need to know the Object Model adhered to by an Object Manager (23). A request to any object in the system (11) is independent of the Object Model of the sought object's Object Manager (23). A generic interface enables new Object Managers (23) reflecting new Object Models to be easily added to the system (11). Availability of the target object (13) is independent of association of the target object (13) with a process (39) at the time the request was issued. Deactivation of processes (39) is automatically accomplished in response to the system (11) needing resources.

☐ 3. Document ID: DE 3140059 A1

L16: Entry 3 of 23 File: EPAB Apr 28, 1983

PUB-NO: DE003140059A1
DOCUMENT-IDENTIFIER: DE 3140059 A1
TITLE: Rolling mill spindle

PUBN-DATE: April 28, 1983

INVENTOR-INFORMATION:

NAME	COUNTRY
GRUDEV, ALEKSANDR PETROVIC	SU
KOMAROV, ALEKSANDR NIKOLAEVIC	SU
DANCENKO, VALENTIN NIKOLEAVIC	SU
ANIKEENKO, IGOR NIKOLAEVIC	SU
UKRAINETS, MICHAIL LOGVINOVIC	SU
TRETYAK, NIKOLAI IVANOVIC	SU
KOSELEVIC, VIKTOR MICHAILOVIC	SU

US-CL-CURRENT: 72/249
INT-CL (IPC): B21B 31/16
EUR-CL (EPC): B21B035/14

ABSTRACT:

CHG DATE=19990617 STATUS=O> The rolling mill spindle is an essential part of the rolling mill. It contains a cylindrical sleeve (1) with recesses (5) on the inner surface, a shaft (2) coaxial to the sleeve (1), and rolling elements (4). Longitudinal grooves (6) are formed on the outer surface of the stub (3) of the shaft (2) which projects into the sleeve (1). A respective rolling element (4) is arranged in the recess (5) and in the groove (6). According to the invention, the rolling mill spindle is additionally provided with a union nut (7) which forms a threaded connection with the central part (8) of the shaft (2) and is fixed in the sleeve (1); the grooves (6) on the stub (3) of the shaft (2) are formed obliquely to the generatrix of this stub (3) of the shaft (2) and the rolling elements (4) are locked in the axial direction of the sleeve (1) and the shaft (2). The rolling mill spindle can be used to adjust the rolls when rolling products of variable cross-section.

☐ 4. Document ID: NNRD440166

L16: Entry 4 of 23 File: TDBD Dec 1, 2000

TDB-ACC-NO: NNRD440166

DISCLOSURE TITLE: Message Definition Language

PUBLICATION-DATA:
IBM technical Disclosure Bulletin, December 2000, UK

ISSUE NUMBER: 440
PAGE NUMBER: 2219

DISCLOSURE TEXT:

Current generation messaging systems or Message-Oriented-Middleware (MOM) provide facilities for reliable communication of messages from one system to another, for putting messages on queues, getting messages from queues, publishing and subscribing to topics and so on. A weakness these systems have in common is that they have nothing to say about the content or format of the messages that they transmit - this information is typically thought to be application dependent. For heterogeneous multi-platform, multi-language inter-operability MOM is lacking in the following areas: * How to construct arbitrary messages in a program written in language A, and receive and interpret them in another program written in language B. (For example, how can a JavaTM (trademark of Sun Microsystems, Inc.) program exchange messages with a COBOL application?) * How to construct messages on one platform, and receive and interpret them on another platform to accommodate character set encodings, numeric type-sizes and endian-ness). * How to integrate quality of service / message context information with a message without impacting application design, separation of systems programming and applications programming concerns. * Management and maintainability of message formats used by applications.

☐ 5. Document ID: NB9402461

L16: Entry 5 of 23 File: TDBD Feb 1, 1994

TDB-ACC-NO: NB9402461

DISCLOSURE TITLE: Automatically Adding Function Prefixes to C/C++ Method Functions for System Object Model Classes

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, February 1994, US

VOLUME NUMBER: 37
ISSUE NUMBER: 2B
PAGE NUMBER: 461 - 462

DISCLOSURE TEXT:

This document contains drawings, formulas, and/or symbols that will not appear on line. Request hardcopy from ITIRC for complete article. - Disclosed is a programming technique for automatically adding function prefixes to the prototypes of C or C++ method functions implementing the methods of SOM classes. - System Object Model (SOM) is a technology for object-oriented programming whereby the programmer specifies the interface to a class of objects in one language (Interface Definition Language, or IDL) and implements the class in his/her preferred programming language (e.g., C or C++). The SOM Compiler generates from the IDL interface specification a template implementation file. This template implementation file is a C or C++ module consisting of "stub" method functions for each of the class's methods. The programmer adds application-specific code to each of the method functions in the template, yielding a complete class implementation. - When specifying the interface to a SOM class in SOM IDL (SOM's implementation of the IDL language), the programmer has the option of specifying a "function prefix" for the class's method functions. If the programmer specifies a function prefix, then as the SOM Compiler generates stub method functions in the template implementation file, the SOM Compiler will construct method function names by prepending the specified prefix to the corresponding method name. If no function prefix is specified, then the SOM Compiler uses the method name as the method function name in the implementation template it generates. The function-prefix facility is useful for avoiding name collisions among the method functions of different SOM classes. - In many cases, a programmer will construct an initial IDL specification for a class without specifying a function prefix, generate a template implementation file using the SOM Compiler from that specification, fill in the stub method functions with application code, and then determine that is it necessary to begin using a function prefix to avoid name collisions. Previously, the programmer's only option was to manually edit every method function in the implementation file and add the new function prefix to the name of every method function. - The invention described here automatically updates the method functions in the implementation file so that all existing method functions have the new function prefix prepended to their names. This is done by the SOM Compiler as it updates the implementation file, adding new method functions for any new methods added to the class and updating existing method function prototypes to match changes in the signatures of previously existing methods. - The invention presented here consists of an algorithm for automatically adding function prefixes to the names of method functions in a C or C++ implementation file for a SOM class. The basic approach is as follows. When the developer invokes the SOM Compiler to update a C or C++ implementation file (update it to match changes in the IDL specification of a class's interface), the user can specify an "addprefixes" flag. The "addprefixes" flag informs the SOM Compiler that, whereas the IDL specification previously did not have a function prefix specified, the current IDL specification does include a function prefix, and that the user wishes the SOM Compiler to update the implementation file to include the specified prefix. When the user specifies the "addprefixes" flag, the SOM Compiler modifies the way that it recognizes matches between method functions in the existing implementation file and the temporary template file that it generates to perform the update. Instead of requiring an exact match between method function names in the two files, the SOM Compiler considers a method function name in the existing implementation file to match a method function name in the new temporary template file if the concatenation of the new prefix and

the old name is equal to the new name (using the "strcmp" C library function). - When an existing method function and a newly generated stub method function are found to be matching by the above criterion, then the SOM Compiler performs the following: First, it transfers the existing method function's prototype to the new implementation file, in comments. (It is preserved so that if an error occurs, the previous prototype will be available.) Second, it adds a comment to the new implementation file stating that a change has been made. Third, it constructs a new method function prototype, by combining the new method function name (which includes the new function prefix), the new method function's parameter types, and the old method function's parameter names. (If, however, the number of parameters has changed or the type of a parameter has changed since the last time the SOM Compiler updated the implementation file, the new method function's parameter names are retained). The SOM Compiler then transfers the resulting method function prototype to the new implementation file, followed by the method function body found in the existing implementation file. - Upon completion, the new implementation file contains all the existing method functions, with the specified function prefix prepended to the name of each in its prototype. - The technique described here makes it much easier for SOM class developers to incrementally develop SOM class libraries. Developers can begin development without consideration for details such as what the function prefix should be; these decisions are best left to a later point in the development cycle, when the complexities of the class library under development are more fully understood. At that point, the function prefix can be added to the class's SOM IDL specification and the SOM Compiler will automatically revise the existing implementation file accordingly. No manual effort is required, other than specifying the new function prefix in SOM IDL, and that specification is only needed once per class rather than once per method. Under the previous release of SOM, the developer was required to manually revise the method function for all of the class's methods. This activity could be quite time-consuming, especially for classes with a large number of methods (and those are precisely the classes for which the function prefix facility is most useful).

6. Document ID: NN9204340

L16: Entry 6 of 23 File: TDBD Apr 1, 1992

TDB-ACC-NO: NN9204340

DISCLOSURE TITLE: Stimulating Shared Buffer Communication in a Distributed Processing Environment.

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, April 1992, US

VOLUME NUMBER: 34
ISSUE NUMBER: 11
PAGE NUMBER: 340 - 350

DISCLOSURE TEXT:

- A sequential (single-thread) computer program is partitioned into several partitions in order to be executed distributively: each partition is to run on a different machine. The partitions are generated by a program partitioning tool (PPT) and use an underlying remote procedure call (RPC) mechanism to communicate at the application level. When an RPC is issued, the data to send and receive within the RPC protocol are written out and read in to/from a communication buffer. The buffer

is "shared" between the two partitions involved in the RPC transaction. The problem is how to implement the RPC mechanism so that it will have the sense that it is using a shared buffer -- thus hide the fact that the buffer is actually copied between the disparate memories of the two processes. Since the fact that two independent processes are involved is not hidden (the RPC mechanism dictates that), synchronization events occur at specific points in the RPC protocol. - An RPC transaction occurs between two partitions when code in one partition issues a call to a routine which resides in a different partition. The partitioning tools (PPT) generate_stubs to help implement the RPC. There is a client stub which is a module residing within the client partition (Note 1), and a server stub, which is a module re siding within the server partition (Note 2). An RPC begins to execute as a local call to a procedure in the client stub. There, the RPC protocol starts. This protocol distinguishes in an RPC transaction two parts: A call transaction and a reply transaction. Between these two events, the server partition may take on the role of a client when its code calls procedures in other partitions (maybe even in the partition which performed the previous RPC to this one). A special code layer called agent handles the proper binding information which supports these back and forth calls. (The agent is not in the scope of this report.) Each RPC transaction part (call or reply) includes transfer of control information to the agent so it can resolve the destination partition for the call, and date (parameters) to be transferred (marshalled) across. Parameters are marshalled using a mechanism called exchange data representation (XDR) which is based on SUN* RPC. This method requires that the encoded version of the parameters be written out (or read in) to/from a buffer or a communication package. When using shared memory communication, the XDR mechanism writes and reads parameters to/from a buffer. A circular buffer is an object (Note 3) which permits the communicating partners continuous read/ write. - When the two partitions reside on different machines which do not have shared memory, the situation can be simulated while also allowing to control the flow of data across the machines in an attempt to optimize on the specific characteristics of the underlying communication method. - Partitions are identified by their partition number. When an RPC is trapped in a client stub, it can recover the number assigned to the target partition where the server code resides. Thus, all further writes and reads for this RPC transaction are done to the partition corresponding to that number. - The circular buffer mechanism for communication is easily implementable using shared memory whereby the hardware support for shared memory provides the interconnection between the two computers (Note 4). Once a shared buffer is created, the connection is set up and ready. When this mechanism is to be employed for other communication media, an elaboration of the hardware solution on top of the new media is needed. We refer to circular buffers on shared memory as our basic model, and the elaborated solution as the extended model. - Two additional layers below the circular buffer are introduced: common communication interface (CCI) and communications flow control (CFC). CCI provides common access to various communication solutions, while CFC is intended to optimize RPC transactions by overlapping parameter marshalling overhead on the two participating machines, and to mediate between the circular buffer and CCI. - Fig. 1 represents a components view of the runtime environment of a partitioned program and its lower layers in the extended model, including CFC and CCI. - Circular buffers play the role of an independent XDR buffer between the application partitions. CFC and CCI layers communicate in a simple protocol whereby they try to copy one output buffer into its corresponding input buffer. The dotted lines stand for control. As can be seen, the circular buffers are activated by either the XDR layer, or the CFC. Likewise, the CCI with the communication drivers are activated by CFC. - In Fig. 1, we see that in order to communicate with two other partitions (0 and 2), partition 1 has two output buffers but only one common input buffer. IRCULAR BUFFERS IN SHARED MEMORY The circular buffer is used via the following calls: CREATE/DESTROY - Circular buffers are created and destroyed with the following routines: CBUF_HANDLE_CREATE - allocates and initializes the i-th circular buffer handle. - CBUF_HANDLE_DESTROY - frees the buffer space. - WRITE - to write into the circular buffer there are three variations of a routine: CBUF_WRITEF - fails if not enough free space is available on circular buffer. - CBUF_WRITEP - writes only part of data if not enough space is available on circular buffers. - CBUF_WRITE - writes to the buffer with possibly several attempts if the data is large. - READ - to read from the circular buffer into a user area there are three variations of a routine as well: CBUF_READF - fails if not all data is available on circular buffer. - CBUF_READP - reads only as much as there is in the buffer. - CBUF_READ - reads in possibly several attempts if not all requested data is available at once. - MISCELLANEOUS - additional service utilities: CBUF_CHKSPACE - checks amount of free and full space in circular buffer. - CBUF_RESET - resets read/write pointers of circular buffer to empty state. - CBUF_INLINE - get space in circular buffer for inline r/w from XDR routines. This

routine allows one to receive a pointer into the circular buffer, with the promise that some n contiguous bytes can be used for read or write where the pointer points. - Each partition owns one buffer which is used for input into it. In an application made of "n" partitions, each partition has access to "n" buffers, of which one is used read-only, and the rest "n-1" buffers are used write-only as output from that partition, and as input to the other partitions. Each buffer can, therefore, be write-accessed by "n-1" partitions, and read-access by only one partition. The fact that there are multiple writers to a buffer is harmless since the partitioned program is used in a single thread mode, whereby all accesses to the buffers are fully synchronized with the activities on a single stack. Therefore, only two partitions are using a buffer at a time, one of which is the writer, and one is the reader. - Each such activity constitutes one part of an RPC transaction: calling and passing input parameters to the server, or returning from a call and receiving output parameters from the server. In the first case, the server and the client use the buffer owned by the server; the server is the reader and the client is the writer. In the latter case, the server and client use the client's buffer; the server is the writer and the client is the reader. - The circular buffers are a mere method for communication between the partitions. Shared buffers can very well be implemented non-circularly. The circular nature allows the two partners to access the buffer in any random rate. The single-thread synchronization assures that each partition will read only what is intended for it by the corresponding writer. Being shared memory, the buffer can be mutually accessed by the reader and writer at the same time thus allowing read and write time overlap. This is advantageous since these operations are the end result of a lengthy process where transferred data is coded into an exchange data representation (marshalling). In this case, the overlap really saves time. CIRCULAR BUFFERS NOT IN A SHARED MEMORY To simulate shared memory between disparate machines, we duplicate each circular buffer so that the "original" buffer resides in its owner partition (the reader of the buffer). Each of the other partitions maintains an active shadow of the original buffer (Note 5). Whenever a writer-partition writes into a shadow buffer, some mechanism takes care of copying it into the original buffer so it can be read by the reader- partition. This mechanism can work (as described below) in a way that exploits the advantages of circular buffers as explained above, and also makes the best use of the underlying communication media. - The performance goals of this process are to get most of the possible overlap between the writer and reader by shortening the time between the event of writing the first byte of a transaction, and the event of reading this byte. ***** SEE ORIGINAL DOCUMENT ***** A copy transaction is defined as one part of an RPC transaction, which goes in one direction only: from the client to the server, or vice-versa. An intuitive approach to performing an optimized copy transaction would be to keep the communication media as busy as possible, yet not to overload it so that the total transmission time will not be stretched out. This means that as soon as the first bytes are written, they are sent across, and the copy operation will keep sending bytes out as long as the following two conditions hold: 1. There is a minimal number of bytes written and ready to be copied (or we are at the end of the copy transaction). 2. The communication media is not congested and is ready to receive more transmissions. - The second point above requires to sense the back pressure from the communication media. We push forward as long as the media is receptive, and wait when it is not. While we wait for the media to finish up what is in its pipe, more data is written to the buffer and is waiting to be sent, thus making the next-to-follow blocks larger. Due to block-transmission overhead, small blocks are less efficient for transmission than large blocks, yet short blocks are ready to be sent earlier than large blocks since the writer of data into the buffer synthesizes it in a given (slow) pace. - Reviewing again the methods described above, we notice that we try to minimize the delay between writing the first byte on one side and reading it on the other side, while also not adding to the overall service time. This is also the criteria of real-time communication; in our context, we want the virtual time of the reader and the writer to coincide as much as possible in real time. - An alternative solution would be to wait for a larger block to be ready and then send it out. This will give the best media usage, yet not the best service per a given individual copy transaction. - To accommodate for any of these two methods, as well as other methods one may come up with, we need to introduce a control protocol between the writer of a buffer and the copy mechanism. We will describe this protocol later on. SYSTEM STRUCTURE REVISITED The extended model in Fig. 1 can be redrawn as in Fig. 2 where additional components are introduced. Rather than viewing the circular buffer as the main interface to the stubs, we will look at the XDR objects and save some resources by using only two buffers. Each partition in an "n"-partitioned application will need "n" XDR objects; one is an input XDR, and "n-1" are output XDRs. Due to the single-thread application model, we observe that: 1. Only two partitions are engaged in a copy transaction at

any one time. 2. For a copy-in transaction, only the input buffer and input XDR

☐ 7. Document ID: NN9104473

L16: Entry 7 of 23 File: TDBD Apr 1, 1991

TDB-ACC-NO: NN9104473

DISCLOSURE TITLE: Access Scheme for Slotted Unidirectional Bus Configurations.

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, April 1991, US

VOLUME NUMBER: 33
ISSUE NUMBER: 11
PAGE NUMBER: 473 - 475

DISCLOSURE TEXT:

- Disclosed is a slotted-and-tagged unidirectional-bus (Stub), a novel access scheme for slotted unidirectional-bus configurations. Its most salient feature is light-load bus-access time of few packet transmission times while providing the capability of transmitting an entire packet in consecutive slots. Due to this slot-contiguity feature, there is no need for segment labeling or concurrent packet reassembly. - One unidirectional-bus configuration for N nodes is the folded bus shown in Fig. 1 Packets are transmitted on the outbound portion and received on the inbound portion. - The Stub access scheme operates in conjunction with slotted transmission and fixed-sixe slots. The most upstream node ("head" node) sends out an infinite sequence of blocks, each containing a number of slots whose combined space for data equals or exceeds the maximum permissible packet size. Each slot, as shown in Fig. 2, begins with a "busy/empty" bit, which is followed by a "tag" bit and then by a fixed- size tag/segment sield. A slot is "busy" if it contains part of a packet, and "empty" otherwise. - All slots are generated "empty", with the tag/segment field in each slot containing the number of remaining slots in its block. The first slot of each block has its "tag" bit set to one, while the others have it set to zero. When a node has a packet for transmission, 0D063, it does the following: (1) Waits until it encounters an empty slot. (2) If the tag bit of this slot is one and its tag value is equal to or greater than the number of slots required for the packet, transmission commences: Every slot is marked "busy". - Unless the end of block is reached, the "tag" bit of the first slot following the packet is set to one. (This slot is always empty.) Else go to (1). - Requiring that a transmission begin in a slot whose tag bit is set to one is absolutely necessary for Stub to guarantee that a transmission never needs to be repeated due to a packet's being overwritten or aborted. A side benefit of the tagged-slot requirement is that all the transmission in a block are packed together, eliminating intra-block fragmentation. - Note that the protocol is designed such that only empty slots contain slot-count numbers (tag values). Moreover, so long as the maximum number of segments (slots) per packet does not exceed 2** (number of bits per slot), the slot size, as determined by the desired segment size, always suffices to contain the count. Thus, the only intra-slot overhead of Stub is one "busy" bit per slot and one "tag" bit per slot. The "busy" bit is usually required anyhow. - The number of slots in each block must at least equal the number of slots needed to contain a maximum-length packet. Increasing the block size beyond this improves the packet packing efficiency, i.e., average used portion of the block. However, it increases the access delay because of the longer waiting times for the beginning of blocks. -

Stub can easily be operated in conjunction with reservation-based access schemes such as CRMA been1|, which are optimized for heavy load but have light-load access time on the order of round-trip propagation de- lay. This application of Stub would result in high performance across the load range. In this case, the "head" would generate Stub blocks only if there were no pending reservations, and a node would transmit either in a reserved slot or when the Stub criteria are met. It is even possible for a node to make a tentative reservation, transmit in a Stub block if it comes by before the reserved slots, and then free the reserved slots when they come by for use by downstream nodes. - Stub can also be used with dual-bus configurations -2-. - A Simplified Stub Protocol: As in the main Stub protocol, in the absence of outstanding reservations, the "head" generates blocks of slots. The number of slots in each block, however, is exactly equal to the number of segments in the maximum-length packet. The tag bit of the first slot in each block is set to one; the tag bits of all other slots are set to zero. "Busy" bits are not required here. (However, they may be used for robustness.) When a node has a packet ready for transmission, it operates as follows: (1) Waits for a tagged slot (tag bit set to one). (2) Sets the tag bit of the slot to zero and transmits its packet. Note that (1) is satisfied whenever the beginning of an empty block is encountered, and no other node can transmit in the same block since the tag bit is set to zero by the first transmitting node. - Due to the restriction of one packet transmission per block, the packing efficiecy of the simplified Stub protocol can be significantly less than that of the main Stub protocol. - References -1- M. M. Nassehi, "CRMA: An Access Scheme for High-Speed LANs and MANs," Proc. of ICC '90, Atlanta, GA, 1697-1702 (April 1990). -2- M. Fine and F. A. Tobagi, "Demand Assignment Multiple Access Schemes in Broadcast Bus Local Area Networks," IEEE Trans. Commun. C-33, 12, 1130-1159 (December 1984).

☐ 8. Document ID: US 20020059212 A1 JP 2002132739 A

L16: Entry 8 of 23 File: DWPI May 16, 2002

DERWENT-ACC-NO: 2002-454256
DERWENT-WEEK: 200248

TITLE: Stub search loading system for use in Java environment computer system, returns appropriate stub for Java run time environment, to request source client, based on designated stub name and client identifier

INVENTOR: TAKAGI, J

PRIORITY-DATA: 2000JP-0322269 (October 23, 2000)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 20020059212 A1	May 16, 2002		022	G06F007/00
JP 2002132739 A	May 10, 2002		019	G06F015/16

INT-CL (IPC): G06 F 7/00; G06 F 9/54; G06 F 15/16; G06 F 17/30

ABSTRACTED-PUB-NO: US20020059212A
BASIC-ABSTRACT:

NOVELTY - A stub search section (13) of each request source client (1-3) sends a

stub request formed from a stub name and a client identifier, to a server (4). A stub search interface (27) of a server, in response to the stub request, returns the stub appropriate for a Java runtime environment of the request source client, on the basis of the designated stub name and client identifier.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are included for the following:

(1) Stub search loading method;

(2) Stub providing server apparatus;

(3) Stub downloading client apparatus; and

(4) Computer readable recorded medium storing stub search loading processing program.

USE - For use in the Java environment computer system.

ADVANTAGE - Provides appropriate stub to be downloaded from the single server computer to each multiple client computers having runtime environments of different types. Dynamically generates the stub, which need not be prepared on the server side in advance.

DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of the stub search loading system.

Request source clients 1-3

Server 4

Stub search section 13

Search interface 27

☐ 9. Document ID: EP 1202174 A2

L16: Entry 9 of 23 File: DWPI May 2, 2002

DERWENT-ACC-NO: 2002-446128
DERWENT-WEEK: 200248

TITLE: Stub search loading system used in computer, transmits stub appropriate for runtime environment of client computer, based on designated stub name and client identifier

INVENTOR: TAKAGI, J

PRIORITY-DATA: 2000JP-0322069 (October 23, 2000)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
EP 1202174 A2	May 2, 2002	E	024	G06F009/46

INT-CL (IPC): G06 F 9/46

ABSTRACTED-PUB-NO: EP 1202174A
BASIC-ABSTRACT:

NOVELTY - A stub search section (13) of the client computers (1-3) transmits a stub request including a stub name and client identifier, to a server (4). A stub search interface (27) of the server in turn transmits a stub appropriate for a JAVA runtime environment of the client, based on designated stub name and client identifier.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are included for the following:

(1) Stub search loading method;

(2) A server for providing stub necessary in executing remote method invocation to a request source client in response to stub request from client;

(3) A client for downloading stub necessary in executing remote method invocation from server; and

(4) A computer-readable recorded medium storing program for executing stub search loading process.

USE - For loading stub search in computer system.

ADVANTAGE - The client having different types of JAVA runtime environments efficiently downloads an appropriate stub from a server within short time. The need for preparing the stubs corresponding to all configurations beforehand for dynamically generating a stub, is effectively eliminated.

DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of a computer system.

Client computers 1-3

Server 4

Stub search section 13

Stub search interface 27

☐ 10. Document ID: WO 200173547 A2 US 20020004848 A1 AU 200149424 A

L16: Entry 10 of 23 File: DWPI Oct 4, 2001

DERWENT-ACC-NO: 2002-061852
DERWENT-WEEK: 200208

TITLE: System for providing a messaging engine for an enterprise Jave-bean enabled server in order to achieve container managed functionality

INVENTOR: JAYACHANDRAN, R; MAHARAJAN, S ; PACHAIPANDIAN, M ; SHEKHAR, A ; SUDARSHAN, K

PRIORITY-DATA: 2000US-193007P (March 29, 2000), 2000US-193003P (March 29, 2000), 2001US-0815481 (March 23, 2001)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
WO 200173547 A2	October 4, 2001	E	038	G06F009/00
US 20020004848 A1	January 10, 2002		000	G06F009/00
AU 200149424 A	October 8; 2001		000	G06F009/00

INT-CL (IPC): G06 F 9/00; G06 F 9/54; G06 F 15/163

ABSTRACTED-PUB-NO: US20020004848A
BASIC-ABSTRACT:

NOVELTY - A messaging server (26) provides guaranteed message delivery and a client system (24) in the enterprise Jave-bean (EJB) architecture (22) includes a client library that is used to route messages from the client system to the messaging

server and to an EJB enabled server (28). The client library may be part of the messaging engine and the architecture may be implemented by one of more client systems, messaging servers and EJB enabled servers.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are included for an EJB architecture, for a method of establishing asynchronous communication between client systems and an EJB enabled server, for a method of generating a stub and for a method of making an asynchronous call.

USE - Providing a messaging engine for an EJB enabled server.

ADVANTAGE - Achieving container managed asynchronous functionality.

DESCRIPTION OF DRAWING(S) - The drawing is a block diagram of the EJB architecture

Messaging server 26

Client system 24

EJB architecture 22

EJB enabled server 28
ABSTRACTED-PUB-NO:

WO 200173547A EQUIVALENT-ABSTRACTS:

NOVELTY - A messaging server (26) provides guaranteed message delivery and a client system (24) in the enterprise Jave-bean (EJB) architecture (22) includes a client library that is used to route messages from the client system to the messaging server and to an EJB enabled server (28). The client library may be part of the messaging engine and the architecture may be implemented by one of more client systems, messaging servers and EJB enabled servers.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are included for an EJB architecture, for a method of establishing asynchronous communication between client systems and an EJB enabled server, for a method of generating a stub and for a method of making an asynchronous call.

USE - Providing a messaging engine for an EJB enabled server.

ADVANTAGE - Achieving container managed asynchronous functionality.

DESCRIPTION OF DRAWING(S) - The drawing is a block diagram of the EJB architecture

Messaging server 26

Client system 24

EJB architecture 22

EJB enabled server 28

☐ 11. Document ID: JP 2000353093 A

L16: Entry 11 of 23 File: DWPI Dec 19, 2000

DERWENT-ACC-NO: 2001-143442
DERWENT-WEEK: 200115

TITLE: Stub and skeleton delivery method in client-server network environment involves generating stub and skeleton so that they are forwarded in single batch

PRIORITY-DATA: 1999JP-0164818 (June 11, 1999)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
JP 2000353093 A	December 19, 2000		004	G06F009/44

INT-CL (IPC): G06 F 9/44; G06 F 15/16

ABSTRACTED-PUB-NO: JP2000353093A
BASIC-ABSTRACT:

NOVELTY - A stub for providing a communication function and skeleton are generated from a server object so that the generated stub and skeleton are forwarded in a single batch to a client.

USE - In client-server network environment.

ADVANTAGE - Enables to forward stub and skeleton to client automatically.

DESCRIPTION OF DRAWING(S) - The figure shows a block diagram of system assembly.

☐ 12. Document ID: RD 440166 A

L16: Entry 12 of 23 File: DWPI Dec 10, 2000

DERWENT-ACC-NO: 2001-326392
DERWENT-WEEK: 200134

TITLE: Message definition language providing facilities for reliable communication of messages from one system to another, for putting messages on queues, getting messages from queues and publishing and subscribing to topics

PRIORITY-DATA: 2000RD-0440166 (November 20, 2000)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
RD 440166 A	December 10, 2000		003	G06F000/00

INT-CL (IPC): G06 F 0/00

ABSTRACTED-PUB-NO: RD 440166A
BASIC-ABSTRACT:

NOVELTY - The method compiles a MDL file using a MDL compiler to generate stub and optionally skeleton files for the chosen target language. Since the OMG already define standard mapping for IDL to many languages, such as C, C++, Java and Smalltalk, etc., the MDL compiler can take advantage of these standards to build language specific stube for the chosen message.

USE - As a message definition language providing facilities for reliable communication of messages from one system to another, for putting messages on queues, getting messages from queues and publishing and subscribing to topics and so on.

ADVANTAGE - Allows an application developer to define messages in MDL, run the MDL compiler to generate stubs for the target languages, and send messages from any platform to any other, and from any programming language to any other.

☐ 13. Document ID: RD 435091 A

L16: Entry 13 of 23 File: DWPI Jul 10, 2000

DERWENT-ACC-NO: 2000-645511
DERWENT-WEEK: 200062

TITLE: Faster finding of a target address in Java by patching 'call indirect' instructions to be 'call relative' where the call has not been compiled

PRIORITY-DATA: 2000RD-0435091 (June 20, 2000)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
RD 435091 A	July 10, 2000		002	G06F000/00

INT-CL (IPC): G06 F 0/00

ABSTRACTED-PUB-NO: RD 435091A
BASIC-ABSTRACT:

NOVELTY - The method is used to patch 'call indirect' instructions to be 'call relative' in cases where the callee has not been compiled, to avoid the overhead of doing the load of the callee address. If the callee method has not been compiled, a 'call relative' is generated at the stub, while the address of the call instruction is put in a patch list containing the address of call instructions intended for the callee but going to the stub instead. Relative offsets in the compiled list are then modified to be 'call relative' and the just-in-time compiler patches each call instruction that jumps to a stub to instead jump to the callee compiled code, so that the call goes directly to the callee address.

USE - Finding a target address in Java (RTM).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 14. Document ID: JP 2000122859 A

L16: Entry 14 of 23 File: DWPI Apr 28, 2000

DERWENT-ACC-NO: 2000-371392
DERWENT-WEEK: 200032

TITLE: Program development assistance device for computer system generates stub class data relevant to execution level of each interface process with identical external interface

PRIORITY-DATA: 1998JP-0297833 (October 20, 1998)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
JP 2000122859 A	April 28, 2000		011	G06F009/06

INT-CL (IPC): G06 F 9/06; G06 F 9/44

ABSTRACTED-PUB-NO: JP2000122859A
BASIC-ABSTRACT:

NOVELTY - The scheduler demand from a source file, interface definition language or from execution format is received based on command class data (96). A stub class data (92) is generated relevant to the degree of execution of each interface process with same external interface as the class specified by user. A hierarchy classifier data (99) containing data specified by user is generated.

DETAILED DESCRIPTION - An INDEPENDENT CLAIM is also included for program development assistance method.

USE - For assisting program development in computer network.

ADVANTAGE - The target hierarchical data is generated precisely thereby program development is simplified.

DESCRIPTION OF DRAWING(S) - The figure shows structure of program.

Stub class data 92

Command class data 96

Hierarchical data 99

☐ 15. Document ID: US 6510192 B1 JP 10274687 A CN 1197275 A US 6205196 B1

L16: Entry 15 of 23 File: DWPI Jan 21, 2003

DERWENT-ACC-NO: 1998-604310
DERWENT-WEEK: 200309

TITLE: Core structure for boiling water reactor used in electricity generation - has stub plane control rods with length equal to fuel assembly width arranged in centre of channel box

INVENTOR: AOYAMA, M; CHAKI, M ; HAIKAWA, K ; KONDO, T ; MASUHARA, Y ; MORIYA, K ; TAKII, T ; YAMANAKA, A ; YAMASHITA, J

PRIORITY-DATA: 1997JP-0079555 (March 31, 1997)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 6510192 B1	January 21, 2003		000	G21C003/34
JP 10274687 A	October 13, 1998		007	G21C005/00
CN 1197275 A	October 28, 1998		000	G21C003/30
US 6205196 B1	March 20, 2001		000	G21C003/34

INT-CL (IPC): G21 C 3/30; G21 C 3/32; G21 C 3/34; G21 C 5/00; G21 C 7/00; G21 C 7/08; G21 C 7/113

ABSTRACTED-PUB-NO: JP 10274687A
BASIC-ABSTRACT:

The structure includes several channel boxes (1) which surround several fuel assemblies (2), individually. Several control rods (4) with four wings (3) are arranged between the channel box. Several long winged control rods (6) are arranged with the wings extending in four diagonal directions and between the channel boxes, in the right direction of grouping the channel boxes. A stub plane control rod (7) is provided in the centre of the channel box. The length of wings of the stub plane control rods are equal to the width of the fuel assemblies contained in the channel box.

ADVANTAGE - Avoids reduction in control rod performance. Achieves reduction in number of threads of control rod. Enables simplification and rationalisation of control system.
ABSTRACTED-PUB-NO:

US 6205196B EQUIVALENT-ABSTRACTS:

The structure includes several channel boxes (1) which surround several fuel assemblies (2), individually. Several control rods (4) with four wings (3) are arranged between the channel box. Several long winged control rods (6) are arranged with the wings extending in four diagonal directions and between the channel boxes, in the right direction of grouping the channel boxes. A stub plane control rod (7) is provided in the centre of the channel box. The length of wings of the stub plane control rods are equal to the width of the fuel assemblies contained in the channel box.

ADVANTAGE - Avoids reduction in control rod performance. Achieves reduction in number of threads of control rod. Enables simplification and rationalisation of control system.

☐ 16. Document ID: EP 972241 B1 WO 9844414 A1 US 5999988 A EP 972241 A1 JP 2002516006 W

L16: Entry 16 of 23 File: DWPI May 14, 2003

DERWENT-ACC-NO: 1998-542882
DERWENT-WEEK: 200333

TITLE: Method of generating and employing run-time generated stub to reference object in object oriented system - by transforming at run-time information associated with remote object into stub class which represents remote object and implements only those interfaces identified by interface descriptor

INVENTOR: HAMILTON, G; KESSLER, P B ; PELEGRI-LLOPART, E ; RIGGS, R ; WALDO, J H ; WOLLRATH, A M

PRIORITY-DATA: 1997US-0829861 (March 31, 1997)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
EP 972241 B1	May 14, 2003	E	000	G06F009/46
WO 9844414 A1	October 8, 1998	E	065	G06F009/46
US 5999988 A	December 7, 1999		000	G06F009/46
EP 972241 A1	January 19, 2000	E	000	G06F009/46
JP 2002516006 W	May 28, 2002		061	G06F009/46

INT-CL (IPC): G06 F 9/46

ABSTRACTED-PUB-NO: US 5999988A
BASIC-ABSTRACT:

The method involves receiving an object reference to the remote object from the second virtual machine. The object reference has information associated with the remote object. The information includes an interface descriptor and an object handle of the remote object. The object handle identifies the remote object and the interface descriptor identifies an interface of the remote object implemented in the second virtual machine. The information associated with the remote object is transformed at run-time into a stub class. The stub class represents the remote object and implements only those interfaces identified by the interface descriptor and defined by the first virtual machine. The stub class is instantiated and the first virtual machine is provided with an instance associated with the stub class.
ABSTRACTED-PUB-NO:

WO 9844414A EQUIVALENT-ABSTRACTS:

The method involves receiving an object reference to the remote object from the second virtual machine. The object reference has information associated with the remote object. The information includes an interface descriptor and an object handle

of the remote object. The object handle identifies the remote object and the interface descriptor identifies an interface of the remote object implemented in the second virtual machine. The information associated with the remote object is transformed at run-time into a stub class. The stub class represents the remote object and implements only those interfaces identified by the interface descriptor and defined by the first virtual machine. The stub class is instantiated and the first virtual machine is provided with an instance associated with the stub class.

☐ 17. Document ID: JP 10049356 A US 5974251 A

L16: Entry 17 of 23 File: DWPI Feb 20, 1998

DERWENT-ACC-NO: 1998-198116
DERWENT-WEEK: 199952

TITLE: Media flow control system - has generator which converts flow definition program to media program that controls input or display of each medium

INVENTOR: OHTA, S; SHIOZAWA, H ; SOHMA, K ; SUEHIRO, R ; YOSHIKAWA, M

PRIORITY-DATA: 1996JP-0216924 (July 31, 1996)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
JP 10049356 A	February 20, 1998		024	G06F009/06
US 5974251 A	October 26, 1999		000	G06F009/445

INT-CL (IPC): G06 F 9/06; G06 F 9/445

ABSTRACTED-PUB-NO: JP 10049356A
BASIC-ABSTRACT:

The system has a flow definition unit which performs batch of individual flow definition (112). The individual flow definition defines the flow which is different for every media. A flow definition program (12) which converts the common flow definition (111) and the individual flow definition to a flow definition for every program, is stored in the flow definition memory units (131-133).

A generator (14) converts the stored flow definition data to a media program which controls the input or display of each media. The media program is stored in the program storage units (151-153).

ADVANTAGE - Facilitates description of common flow depending on media. Enables automatic generation of stub area during test. Simplifies media flow control using mark-up language.
ABSTRACTED-PUB-NO:

US 5974251A EQUIVALENT-ABSTRACTS:

The system has a flow definition unit which performs batch of individual flow definition (112). The individual flow definition defines the flow which is different for every media. A flow definition program (12) which converts the common flow definition (111) and the individual flow definition to a flow definition for every program, is stored in the flow definition memory units (131-133).

A generator (14) converts the stored flow definition data to a media program which controls the input or display of each media. The media program is stored in the program storage units (151-153).

ADVANTAGE - Facilitates description of common flow depending on media. Enables automatic generation of stub area during test. Simplifies media flow control using mark-up language.

☐ 18. Document ID: JP 09185517 A

L16: Entry 18 of 23 File: DWPI Jul 15, 1997

DERWENT-ACC-NO: 1997-412220
DERWENT-WEEK: 199738

TITLE: Distributed object management system for computer - continues process of formation component element program of class example by terminating stub and trouble generated in stub memory area

PRIORITY-DATA: 1995JP-0342217 (December 28, 1995)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
JP 09185517 A	July 15, 1997		009	G06F009/46

INT-CL (IPC): G06 F 9/44; G06 F 9/46

ABSTRACTED-PUB-NO: JP 09185517A
BASIC-ABSTRACT:

The system includes a memory area which registers whether an example data request element program and an object is independent into a class memory area attribute for every class. If the class memory area attribute is independent, a class example is formed in response to the demand for the class example formation. A stub which is a program that corresponds to the class is read from a stub memory area. An element and an instance identifier are stored in the stub memory area. When the class example procedure execution is requested, a message is transmitted to the stub of the memory area to perform the procedure execution to the example shown by the example identifier.

A response message is received and a result is returned. When the class example deletion is requested, the message is transmitted to the memory area stub to delete the example shown by the example identifier. A response message is received and the result is returned. The stub is terminated and the trouble generated in the stub memory area is also terminated. The formation component element program of the class example is continued.

ADVANTAGE - Prevents object user data from being lost.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Clip Img | Image

☐ 19. Document ID: EP 546684 A2 EP 546684 A3 US 5421016 A

L16: Entry 19 of 23 File: DWPI Jun 16, 1993

DERWENT-ACC-NO: 1993-190055
DERWENT-WEEK: 199324

TITLE: Object orientated data processing system generating stubs - has static methods stored in memory with compiler generating class definitions with function unit transferring control without changing binary K

INVENTOR: CONNER, M H; COSKUN, N ; MARTIN, A R ; RAPER, L K

PRIORITY-DATA: 1991US-0805778 (December 12, 1991)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
EP 546684 A2	June 16, 1993	E	044	G06F009/44
EP 546684 A3	December 8, 1993		000	G06F009/44
US 5421016 A	May 30, 1995		027	G06F009/44

INT-CL (IPC): G06F 9/44

ABSTRACTED-PUB-NO: EP 546684A
BASIC-ABSTRACT:

The processing system includes a storage part for storing one or more static methods and a compiler for generating class definitions and redispatch stubs for one or more static methods and a dispatch function part for processing the redn. patch stub.

The function part also transfers control to the static method associated with the redispatch stub. Any binary application can dynamically invoke any of the one or more static methods without changing the application binary.

The redispatch stub is a short sequence of calling sequence instructions of a particular one or more static methods. The storage part is random access memory. The system includes a data structure for storing the stubs or methods.

USE/ADVANTAGE - for object-oriented programming. Can change methods without corrupting execution of application.
ABSTRACTED-PUB-NO:

US 5421016A EQUIVALENT-ABSTRACTS:

A SOM compiler generates class definitions and generates a redispatch stub for each method defined in a class. A redispatch stub is a short sequence of instructions with an identical calling sequence as its corresponding method. This gives the class' dispatch enough information to determine the correct method procedure in a dynamic manner.

The dispatch function employs the redispatch stub to call the appropriate method procedure and return any return value to the calling application via the redispatch stub. Redispatch stubs allow a class with a definition that can vary at runtime to be used by an application that was designed for statically defined classes.

ADVANTAGE - Allows application designed to use static method calls to manipulate objects whose methods are only available through dynamic calls without modifying binary image of application.

☐ 20. Document ID: EP 501610 A2 DE 69228621 E EP 501610 A3 US 5475817 A EP 501610 B1

L16: Entry 20 of 23 File: DWPI Sep 2, 1992

DERWENT-ACC-NO: 1992-294059
DERWENT-WEEK: 199922

TITLE: Object oriented distributed computing system - has processor calling location service within automatically generated stubs in response to request for service by particular object

INVENTOR: ARNOLD, K C; CANNON, M J ; ERDOS, M ; HARRISON, B D ; HOFFMAN, D J ; MCBRIDE, B W ; ROBINSON, D B ; SEABORNE, A F ; SHOWMAN, P S ; SMITH, L D ; WALDO, J A ; ERDOS, M E ; WALDO, J H

PRIORITY-DATA: 1991US-0[redacted]0478 (February 25, 1991), 1993U[redacted]0159764 (November 30, 1993)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
EP 501610 A2	September 2, 1992	E	020	G06F009/44
DE 69228621 E	April 22, 1999		000	G06F009/44
EP 501610 A3	June 9, 1993		000	G06F009/44
US 5475817 A	December 12, 1995		016	G06F015/16
EP 501610 B1	March 17, 1999	E	000	G06F009/44

INT-CL (IPC): G06F 9/44; G06F 15/16

ABSTRACTED-PUB-NO: EP 501610A
BASIC-ABSTRACT:

An object oriented computing system 911) has a number of computers (37) and processes (39) and a member of objects (13) for performing respective operations in different. A management apparatus includes a location server (21) for locating objects in the computing system (11) on behalf of requesters in a manner than is independent of register knowledge of the location service.

The locations service is automatically called on behalf of a requester in one process transmitting a request is a target object.

ADVANTAGE - Does not require clients of services to be coded so as to know or acquire locaitons of service providers. Recognises objects that conform to various object models.
ABSTRACTED-PUB-NO:

EP 501610B EQUIVALENT-ABSTRACTS:

An object oriented computing system 911) has a number of computers (37) and processes (39) and a member of objects (13) for performing respective operations in different. A management apparatus includes a location server (21) for locating objects in the computing system (11) on behalf of requesters in a manner than is independent of register knowledge of the location service.

The locations service is automatically called on behalf of a requester in one process transmitting a request is a target object.

ADVANTAGE - Does not require clients of services to be coded so as to know or acquire locaitons of service providers. Recognises objects that conform to various object models.

US 5475817A

An object oriented distributed computing system supporting a plurality of object models, each object model being defined by the way the persistent state of the model's objects is managed and the way operations on the model's objects are mapped into code to perform the operations, said mapping into code being the execution model of the model's objects, comprising:

a first computer,

a second computer,

a manager of object managers located on the first computer,

a manager of object managers located on the second computer,

a plurality of object managers located on the first computer, each based on a different object model,

at least one object manager located on the second computer, based on an object model,

objects of one object model, including a persistent state and an execution model, communicating operation requests to objects of a different object model by making calls in the system, each request including (i) an object identifier indicating a target object and (ii) an indication of an operation desired to be performed by the target object, each object manager supporting operation and existence of objects associated with that object manager according to the object model of that object manager, and each manager of object managers communicating with respective associated object managers in a manner free of preprogrammed code in the operation call, of the respective object models of the object managers to locate objects to which operation requests are to be communicated, such that a request by a first object managed by a first object manager on the first computer to perform an operation on a second object managed by a second object manager on the same computer is communicated from the first object to the manager of object managers on the first computer and then to the second object manager, the manager of object managers on the first computer being responsive to the first object and (i) based On the second objects identifier but independent of the second object type and independent of object model of the second object manager, determines whether the second object is managed by the second object manager and (ii) communicates the request to the second object manager, the second object manager, in accordance with its object model, obtains the persistent state and execution model of the second object and therefrom activates the second object and delivers the request to the second object, and

a request by the first object managed by the first object manager on the first computer to perform an operation on a third object managed by a third object manager on the second computer is communicated from the first object to the manager of object managers on the second computer and then to the third object manager, the manager of object managers on the second computer being responsive to the first object and (i) based on the third objects identifier but independent of the third object type and independent of object model of the third object manager, determines whether the third object is managed by the third object manager and (ii) communicates the request to the third object manager, the third object manager, in accordance with its object model, obtains the persistent state and execution model of the third object and therefrom activates the third object and delivers the request to the third object.